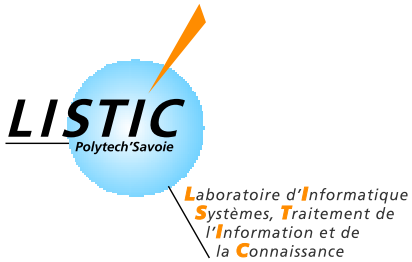
Integrating OPC data into GSN infrastructures

Eric Benoit, Marc Philippe Huget, Patrice Moreaux, Olivier Passalacqua

Integrating OPC Data into GSN Infrastructures

Design and implementation of an OPC-GSN wrapper

É. Benoit, M.-P. Huget, P. Moreaux, O. Passalacqua



BP 80439
74944 ANNECY LE VIEUX Cedex
FRANCE

Research Report 2007-10-01
october 2007

Résumé

Nous présentons la conception et le développement d'un composant logiciel d'interface entre des données au format OLE for Process Control (OPC) et l'infrastructure Global Sensor Network (GSN) de gestion de données de capteurs. Cette interface, dénommée *wrapper* dans le contexte GSN, communique en mode *Data Access* avec un serveur OPC et convertit les données reçues au format interne de GSN, selon différents modes temporels.

Ce travail est réalisé dans le cadre d'une thèse sur le contrôle des systèmes répartis de fusion d'information. Le composant que nous avons développé permet d'injecter des données OPC, comme des mesures ou des information d'état d'un processus industriel, dans un système réparti de fusion d'informations implanté sur une infrastructure GSN.

Ce composant se comporte comme un client du serveur OPC. Programmé en Java et basé sur le projet Openscada Utgard, il peut être déployé sur tout noeud de calcul acceptant une machine virtuelle Java. Les tests que nous avons réalisés montrent la conformité du composant aux spécifications *Data Access* 2.05a du standard OPC ainsi que la disponibilité des différents modes temporels.

Abstract

We present the design and the implementation of an interface software component between OLE for Process Control (OPC) formatted data and the Global Sensor Network (GSN) framework for management of data from sensors. This interface, named wrapper in the GSN context, communicates in Data Access mode with an OPC server and converts the received data to the internal GSN format, according to several temporal modes.

This work is realised in the context of a Ph.D. Thesis about the control of distributed information fusion systems. The component we have developed allows injection of OPC data - like measurements or industrial processes states information - into a distributed information fusion system deployed in a GSN framework.

The component behaves as a client of the OPC server. Developed in Java and based on the Opensaca Utgard, it can be deployed on any computation node supporting a Java virtual machine. The experiments we did show the component conformity according to the Data Access 2.05a specification of the OPC standard and to the temporal modes.

Contents

List of Figures

List of Tables

1 Introduction

Last decade has seen more and more applications based on information fusion. These applications range from military domain, such as target detection, to everyday life with house equipments. Morover industrial workshops use an increasing number of various sensors for machine control as well as for information processes. It is then interesting to allow these information systems to benefit from information fusion capabilities. In this perspective, we will have to interconnect industrial data systems with Information Fusion Systems (IFS).

On one side we have the area of industrial data systems, where the OLE for Process Control standard (OPC) [OPC] is largely deployed since the mid'90s and this standard is continuously evolving. OPC allows interconnection of data sources and data sink through communication managers called OPC *servers*. Data sources and sink termed as OPC *clients* may be industrial equipments, measure devices, alarm buttons, etc. Although last version of the OPC standard introduces communication based on Intranet and Internet protocols, a large part of installed OPC networks use the COM/DCOM interaction model developed in Microsoft Windows environments.

On the another side, there exist a lot of IFS, either academic or commercial. They address very different kinds of data sources ranging from low power sensors to huge databases. Software and hardware IFS vary also largely. Some systems are very specialized, some are able to be deployed on many kinds of processing nodes. In the perspective of our research activities, we are using the Global Sensor Network (GSN)[Ali07] as an experimental IFS. GSN is a research project of the LSIR research laboratory (EPFL, Lausanne, Switzerland) developed in Java, allowing us to quickly deploy a simple IFS on several computers. GSN is already equipped with several input converters (termed "wrappers") for connection of various data sources to a GSN infrastructure. However, there exist no wrapper for connecting OPC data sources to GSN.

We present in this report, the design and the implementation of a GSN wrapper for connection of OPC data sources to a GSN node.

There exist several approaches to connect OPC servers to GSN. Since we will mostly deal with installed OPC systems, we have chosen not to impact the OPC configuration. Consequently, our wrapper behaves exactly as an OPC client getting data from one OPC server in a COM-DCOM context. It then formats these data in the GSN data format (termed *StreamElement* in the GSN context). Implementation of the wrapper is written in Java and is based the open-source Openscada Utgard project [CCJ06].

The organization of the report is the following. Section 2 presents the main features and the software architecture of the GSN system. Section 3 recalls the OPC standard and provides a simple illustrative example. We expose the design and the implementation of our wrapper in Section 4. We have experimented our OPC-GSN wrapper and results are presented in Section 5. Section 6 summarizes our work and proposes future work.

2 GSN

2.1 Overview

The Global Sensor Network project (GSN) [KMA07a, KMA07b] is a middleware supported by several teams of researchers and developers. Its purpose is to get data from various sources such as sensor networks, and to process them through functional nodes. A general GSN system is made of several nodes we call *GSN core* (nodes) and sensor networks connected to these GSN cores. Several GSN cores may run on a computer or may be interconnected through an underlying

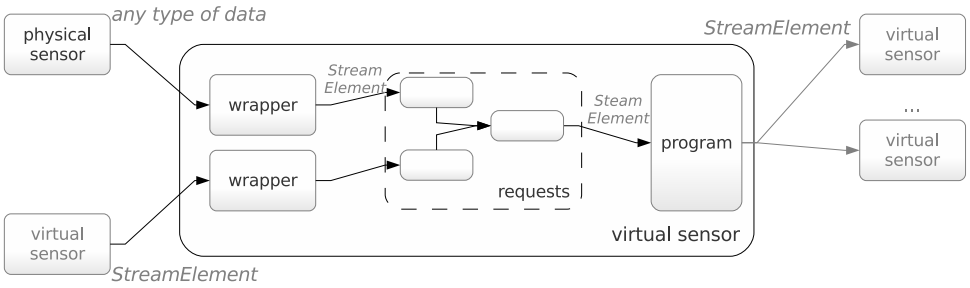


Figure 1: Virtual sensor structure

GSN data type	Description
CHAR	A string of fixed length
VARCHAR	A string of variable length
TINYINT	An 8-bit signed integer value (range: -128 to 127)
SMALLINT	A 16-bit signed integer value (range: -32768 to 32767)
INTEGER	A 32-bit signed integer value (range: -2147483648 to 2147483647)
BIGINT	A 64-bit signed integer value (range: -9223372036854775808 to 9223372036854775807)
DOUBLE	A 64-bit precision floating point value (identical to the Java double type)
BINARY	A variable sized binary object

Table 1: GSN data types

network of computers. A GSN core is a Java program running inside one Java Virtual machine (JVM). Each core has a default data-base system used to store and to process information. A detailed presentation of GSN is given in [Ali07].

Wrapper, virtual sensor and GSN core A GSN core manages a set of Virtual Sensors (VSs), usually partially connected. VSs are the main elements of the GSN information process. A VS (see Figure 1) selects information received by its sources, termed *wrappers* in GSN, aggregates some of these selected information, optionally applies a specific processing to this aggregate and finally transmits the result to another VS or to an entity external to GSN. Wrappers are the links between information sources outside the VS and the VS. Sources may be sensors networks (outside of GSN) or output of other VSs. A wrapper is in charge of the communication with its sources (written in Java, it can communicate with all sources reachable from a Java class, ranging from a serial connection to a Web service). It formats its output data in GSN compatible data structures termed *StreamElements*. Information selection from each wrapper is carried out by a SQL-like query (we call it a *wrapper request*) and result is stored into a data-base. Following information aggregation is then processed also with SQL-like queries (let us call it the *global request*) on these results possibly from several wrapper output selections. Possible specific final processing and result output is achieved by an instance of a Java class defined by the designer of the VS. This Java class must implements a Java interface given by GSN. Alternatively to sending output result to another VS, it can display it on various user interfaces such as a Web browser.

GSN StreamElement StreamElements are data units processed by GSN. The structure of a StreamElement is the following:

- An array of strings with the name, the type and the description of each produced element,
- An array with the value of the produced element,
- A timestamp.

For instance a StreamElement recording temperature and pressure of a given room will be:

temperature	INTEGER	temperature in room B206
pressure	INTEGER	pressure in room C120

25
1024

StreamElement production time (in ms)

Data types handled by GSN are summarized in Table 1

Let us emphasize that the resulting data flow in GSN is a *flow of (discrete time) data elements* (the StreamElements) even if the input of a wrapper may be sustained data.

Example of GSN application Let us describe a VS processing data from two cameras each of them producing a couple (picture,compression rate) every minute. We use two identical wrappers receiving these couples, one for each camera. Each wrapper request selects and stores only pictures with a compression rate greater than 0.9. Then the global request may select, say the most recent picture among the last two pictures and a Java class could convert this picture from JPEG to BMP type for instance.

VS description Java classes for wrappers and VS must be loadable at the starting time of the GSN core. At startup, a GSN core read the configurations of its VS from XML description files (one for each VS), and load required Java code into its JVM. A VS description file is divided into three parts (wrappers used, wrapper and global requests, Java class) and describes the architecture of the VS, which requests should apply, what types of data are processed and what is the Java class of the VS. It also defines symbolic names for the various data flows managed by the VS and its wrappers.

2.2 Behaviour of GSN components

Interconnections between wrappers and VS require the designers of GSN to define a policy about which VS and wrappers are effectively able to produce data and StreamElements at any time. Although not clearly stated in GSN publications, this implies to identify two logical states for a wrapper or a VS:

- not alive: in this state the wrapper or the VS is either unavailable or no more usable.
- ready: in this state the wrapper or the VS is able to receive data from outside of GSN or StreamElements from other VS and to send StreamElements.

The GSN policy states that if one of the wrappers of a VS v is not alive then v is also not alive. Moreover if a wrapper cannot get data from its source and generates an error caught by the GSN core, then it is not alive. Finally, Java code of wrappers and VS must be loadable at startup time of their GSN core, otherwise they will never be ready.

Dynamic configurations of GSN systems A GSN core is continuously aware of all XML VS description files in a specific directory and it gets at startup, the list of its possible wrappers in a file giving pairs (GSN wrapper name, wrapper Java class name). Java classes of the wrappers must be available at startup. Thus GSN cannot, at this time of writing, handle dynamic injection of new wrappers. However, a GSN core user can force it to start and stop a

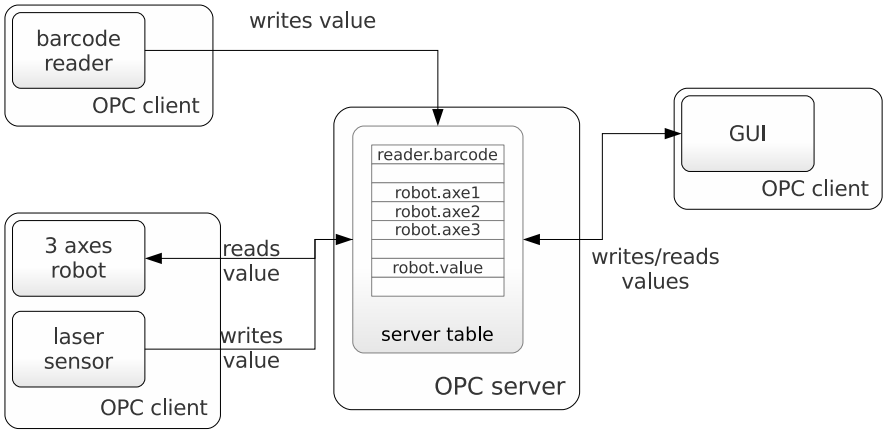


Figure 2: Example of an OPC based system

VS simply by putting/removing its XML description file in the dedicated directory. We plan to extend capabilities of GSN cores to allow us for insertion of new wrappers and VS with different configurations at run time.

3 OPC

3.1 Introduction

Object linking and embedding for Process Control (OPC) is a set of specifications initially written by the WinSEM (Windows for Science Engineering and Manufacturing) group to standardize interfaces between devices and applications in the context of workshops and Microsoft Windows Operating Systems with COM-DCOM protocols. Before OPC, data exchanges were achieved using the previous Microsoft Windows standard Dynamic Data Exchange (DDE), but the limitations of this standard were quickly reached. The OPC standard defines objects, interfaces and methods to control and facilitate the interoperability between applications and devices which are responsible for acquiring data. Information is sent by the devices to one or several OPC servers that group data and expose it to client applications or other devices. These applications can be Supervisory Control and Data Acquisition (SCADA) systems or dedicated clients. An overview of the OPC concepts is discussed in [Li 02].

Since 1996 new OPC specifications are released and updated. Initially, data exchanges between OPC servers and clients were based on the COM-DCOM Windows Microsoft standard. However, interoperability with more and more various devices, not necessary running on a Microsoft Windows operating system environment, leads the OPC foundation to define a new general OPC context and new communication protocols. This last OPC specification [SW06], named *Unified Architecture* (UA), is using a unique Web service interface and all operations such as Data Access, Alarm and Even and others (see below for details) have been redefined in this context.

3.2 Illustrative example

Let us consider a very simple workshop where elements of a body car move on a travelling belt. Each element is on a pallet identified at the bottom part by a barcode. A 3 axes robot must carry out some measurements with a laser sensor on the element and the set of measurement positions are computed based on the code of the element. This process may be implemented (figure 2) through:

- An OPC server storing barcodes provided by the bar code reader,

- A bar code reader sending codes to the OPC server,
- A computer computing the set of measurement positions for each bar code,
- A robot reading its measurement positions on the OPC server and sending the measurements to this server.
- A Graphical User Interface (GUI) running on the computer, to display measurements and to alert the user if something goes wrong.

3.3 Communication modes in OPC systems

OPC servers use tables where the last received values are stored and can be read by OPC clients, which ask for values present at dedicated addresses in the table. OPC specifications (see the OPC foundation Web site [OPC]) define several modes of communication.

Data access The first one introduced was the *Data Access* (DA) specification, which is a request/response mode. In this mode, the OPC server can be seen as a shared memory where the OPC clients can read or write values, and the role of the server is to manage concurrent accesses to data. This is the most used communication mode in installed OPC systems and it is well adapted to measurement extraction from a workshop for processing outside the workshop. Our OPC-GSN wrapper conforms to the DA specification.

Alarm and Event The *Alarm and Event* (AE) specification provides notifications on user's demand. In this case the user has to register a logical expression on the server (such as "temperature higher than 50"). The server will eventually send the values (the temperature or the alarm signal) to the registered client when the expression is satisfied. Since our information fusion system is fully managed by GSN, we do not take into account events generated outside of GSN so we don't integrate the AE specification in our OPC-GSN wrapper.

Batch The *Batch* specification relates to batch processing modes and the IEC 61512-1 standard. We did not implement the Batch specification since we only see the OPC system as a item value based data source.

Data eXchange The *Data eXchange* (DX) specification allows the communications between OPC servers. The protocols used in this specification are internal to OPC systems so that our OPC-GSN wrapper has not to be aware of this mode.

Historical Data Access The *Historical Data Access* (HDA) specification provides the same service as the DA specification but taking into account the time history of the values. Such a mode is not needed in our approach since GSN itself provides all the mechanisms to deal with sequences of values such as search in the past values, aggregation over the history of an item, etc.

Security The *Security* specification specifies how to control client access to the servers in order to protect sensitive information. This specification is not needed in our case for two reasons. In the present work, we assume that we are able to install a GSN core running our OPC-GSN wrapper inside the protected area of the OPC system. Then, GSN supporting high level security mechanisms using network protocol such as HTTPS, this GSN core can be securely connected to other GSN cores. Including security protocols between the OPC server and the OPC-GSN wrapper is scheduled in a future release of our wrapper.

XML-DA The *XML-DA* specification provides flexible, consistent rules and formats for exposing data using XML. We do not integrate this specification in our wrapper since we restrict input data to the types managed by GSN. Thus the possible types are the common SQL ones (see table 1).

Complex Data The *Complex Data* (CD) specification allows the servers to expose and describe complicated data such as binary structures. Here again, this kind of objects are out of the scope of the OPC-GSN wrapper.

Commands The *Commands* specification is a set of interfaces that allows OPC clients and servers to identify, send and monitor control commands on a device. We do not use this specification because we see the OPC system as a data source only. We do not manage control of the OPC server but on the whole system. However, the development of adaptability functionalities with regard to the fusion system on top GSN will probably require to send control commands to the OPC system.

3.4 Item content

An OPC server provides both data item values and information about these values. For instance, in DA mode, an OPC client receives the following attributes for each subscribed item:

- The value of the item (such as a barcode read or a temperature),
- The quality of the value (that can be good, bad or uncertain),
- The error code if any,
- The timestamp of the value which is updated even if the value has not changed.

3.5 COM-DCOM

OPC clients and servers are based on the Microsoft (Distributed) Component Object Model (COM-DCOM) [Ros98], which defines how components interact (in a client/server framework). This standard allows several components alive in a Microsoft Windows operating system (OS) to communicate through system calls processed by the COM-DCOM service of the OS. The components may be in the same process, or in different processes or even on different Windows OS (and computers). The OPC client has to know the DCOM identifier of the OPC server before asking it for values. Note that using the COM-DCOM standard binds OPC systems to Microsoft OS. The new UA architecture will allow OPC systems to spread across non Windows OS.

3.6 OPC simulators

OPC simulators have been developed by companies selling OPC servers and clients to ease OPC application developers tests. These software are usually effective OPC servers configured to allow the user to define a simulation load (items, sources and sinks) in a virtual OPC system and to connect the client under test to this server.

To test our OPC-GSN wrapper we used two simulators: the Matrikon simulator (`http://www.matrikon.com/products/opc.aspx`) and the DSxP simulator (`http://www.dsxp.com`). We have chosen the Matrikon simulator since Matrikon is a very well known OPC solutions provider company. To confirm the compatibility of our wrapper with OPC DA standard, we also tested it with the DSxP simulator. Each simulator has specific restrictions we encountered during tests.

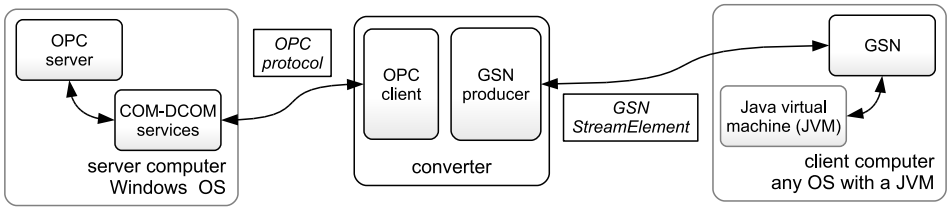


Figure 3: Functional view of the converter

4 OPC integration in GSN

The goal of the OPC-GSN wrapper is to allow a GSN system to get data from an OPC system (see Figure 3). In this first version, we consider the OPC system as a data source only and we do not control the OPC system. Information fusion functions are provided by the GSN system and the OPC server is communicating with the wrapper in the DA mode. Since OPC servers expose data items at any time and GSN infrastructures deal with discrete data flows, the first design problem is to define a timed conversion semantics between OPC data and GSN StreamElement flows. The second main design choice is related to the position of the converter in the software/hardware architecture made of the OPC server and its connected GSN core.

4.1 Timed semantics of data transfer from OPC to GSN

Discrete data flows in GSN are (timed) sequences of StreamElements produced by wrappers and consumed by VS. Since OPC data are available at any time, we are then faced the problem of the production dates of the StreamElements corresponding to OPC items. Moreover, on one hand, the OPC server usually samples its item values with a given rate (period Δt_S) and on the other hand, the converter must update OPC data with requests to the OPC server with another sampling rate (period Δt_U). For what concerns the converter, we can define at least two production modes: either the converter produces one StreamElement every Δt_P time units (*Periodic Production Mode* with period Δt_P, PPM(Δt_P)), or else the converter emits one StreamElement only if the item value (without its timestamp) has changed between to requests to the OPC server (*Change Based Production Mode* with period Δt_U, CBPM(Δt_U)). In PPM, we should have $\Delta t_P = \Delta t_U$ otherwise the converter will produce several StreamElements with the same information ($\Delta t_P < \Delta t_U$) or some values may be lost ($\Delta t_P > \Delta t_U$). Impact of the various relationships between $\Delta t_S, \Delta t_U$ and Δt_P, illustrated in figure 4, are the following. We distinguish two PPM cases and one CBPM case:

- Case 1: PPM with $\Delta t_S = \Delta t_U$. Then, generated StreamElements and OPC item values are synchronized. No OPC item change is "lost".
- Case 2: PPM with $\Delta t_S > \Delta t_U$. The converter produces more SreamElements than required to follow OPC item values changes and several successive StreamElements will carry the same OPC item value.
- Case 3: CBPM. In this case, Δt_P is not defined. Depending on the relationship between Δt_S and Δt_U, the converter may miss some modifications of the OPC item values as in case 2. However, it cannot obviously generate "redundant" StreamElements as in case 3. The figure corresponds to $\Delta t_S > \Delta t_U$ and exactly each OPC item value change generates one StreamElement.

Clearly, we should have $\Delta t_{S,i} \geq \Delta t_U$ for *each* OPC item i converted to be sure not to "lose" values from the OPC server. Thus, the OPC-GSN wrapper user must set $\Delta t_U \leq \min_{i \in I}\{\Delta t_{S,i}\}$, where I is the set of all items converted from a given OPC server. Since a given OPC-GSN

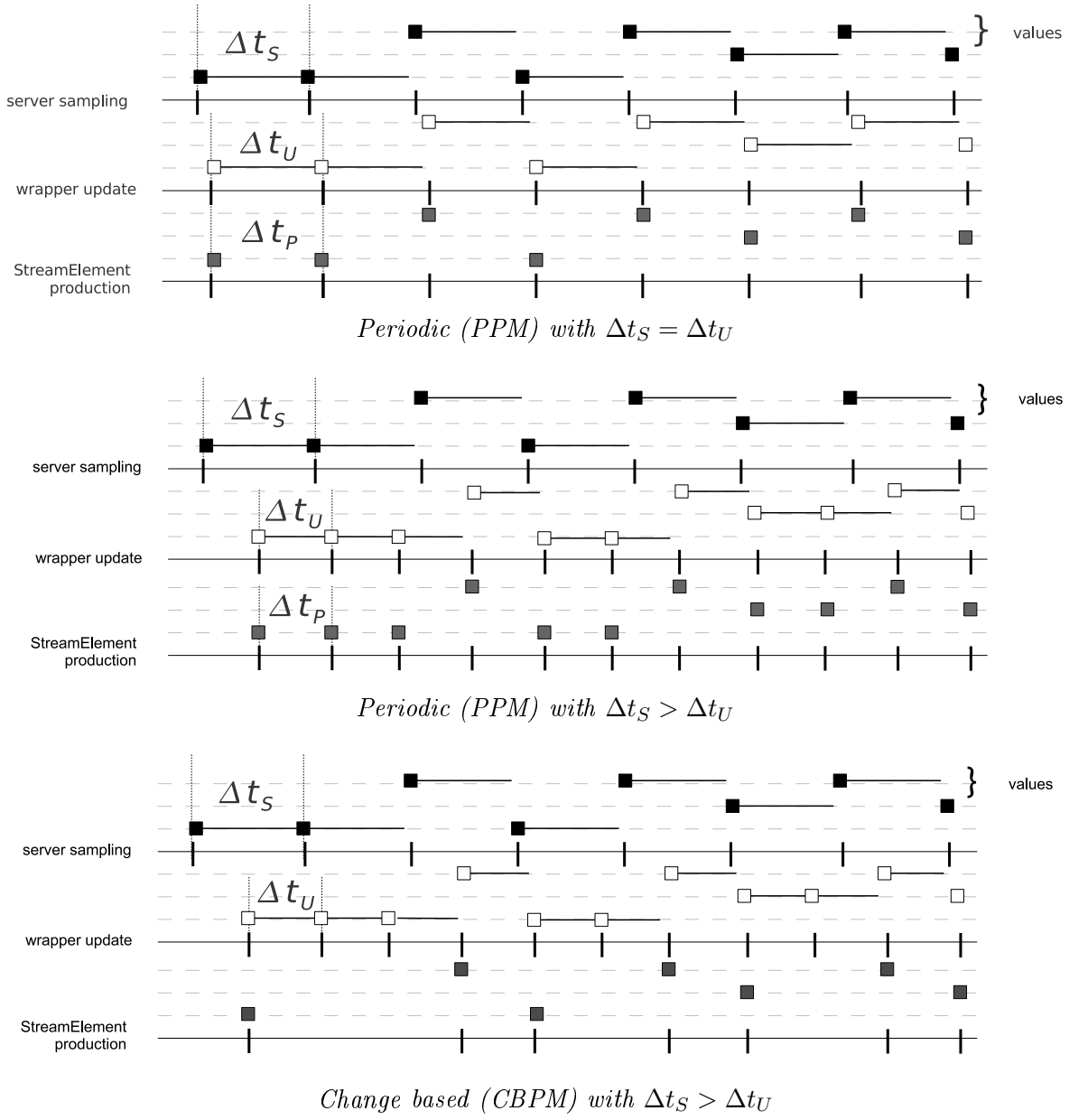


Figure 4: Conversion modes from OPC values to GSN StreamElements

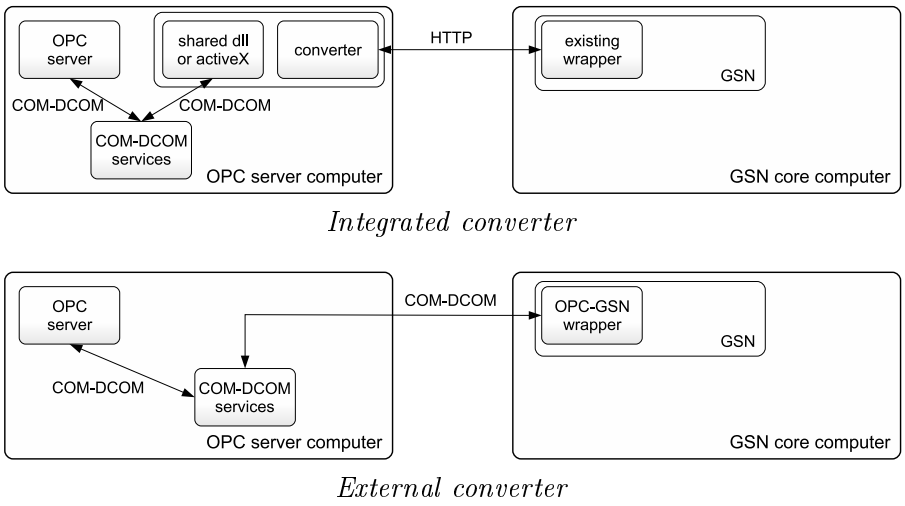


Figure 5: Alternative implementations of the OPC-GSN converter

wrapper is connected to one only OPC server, each Δt_U choice for the same OPC server requires one wrapper.

4.2 Implementation architectures

We have two possibilities for positioning the converter in an OPC-GSN system (see Figure 5):

- Integrated converter: the converter is a software running on the Windows OS of the server (or a Windows OS COM-DCOM connected to the server). This solution requires the converter to generate data over HTTP connection with a GSN core, and to communicate with the OPC server through COM-DCOM services calls.

- External converter: the OPC-GSN converter runs outside the Windows OS of the server and communicates with it through COM-DCOM: it can then be a GSN wrapper.

Our implementation is external. Two reasons motivated us for choosing this solution. First integrating OPC data inside a GSN system does not mean the ability to modify the possibly running OPC system. It could then be forbidden or technically impossible to add a converter on the OPC side of the application. Second, removing the Windows OS constraint for the converter, we are able to provide an OPC-GSN connection running on any system supporting GSN and not only Windows OS.

4.3 OPC-GSN wrapper software

The wrapper conforms to OPC DA 2.05 and converts OPC data to GSN data type as indicated in Table 2. Section A.4.2 describes the output format of the wrapper. The wrapper is implemented in Java (5) and supports two production modes (PPM and CBPM). Figure 6 gives the architecture of the wrapper. Connection to the OPC server is based on the Openscada Utgard project library [CCJ06]. This project provides a Java OPC client and it uses the J-Interop library to manage the COM-DCOM protocol.

The upper left side of Figure 6 shows partially implemented features of the wrapper. Since we plan to use GSN as a Controlled and Adapted Distributed Information Fusion System, the wrapper will be controllable. At the moment, we have implemented the definition of Δt_U and production mode: these parameters are regularly monitored by the wrapper so that *they can be changed at runtime.*

OPC data type	Java data type	GSN data type
short integer	Short	SMALLINT
integer	Integer	INTEGER
single float	Double	DOUBLE
double float	Double	DOUBLE
currency	Double	DOUBLE
date	String	VARCHAR(100)
string	String	VARCHAR(100)
boolean	String	CHAR(5)
byte	Byte	TINYINT
unsigned byte	Short	SMALLINT
unsigned word	Integer	INTEGER
unsigned double word	Long	BIGINT
array of double	String	VARCHAR(1000)
word	Short	SMALLINT
double word	Integer	INTEGER
unsigned short	Integer	INTEGER
unsigned integer	Long	BIGINT
character	Short	SMALLINT

Table 2: Data type mapping between OPC, Java and GSN

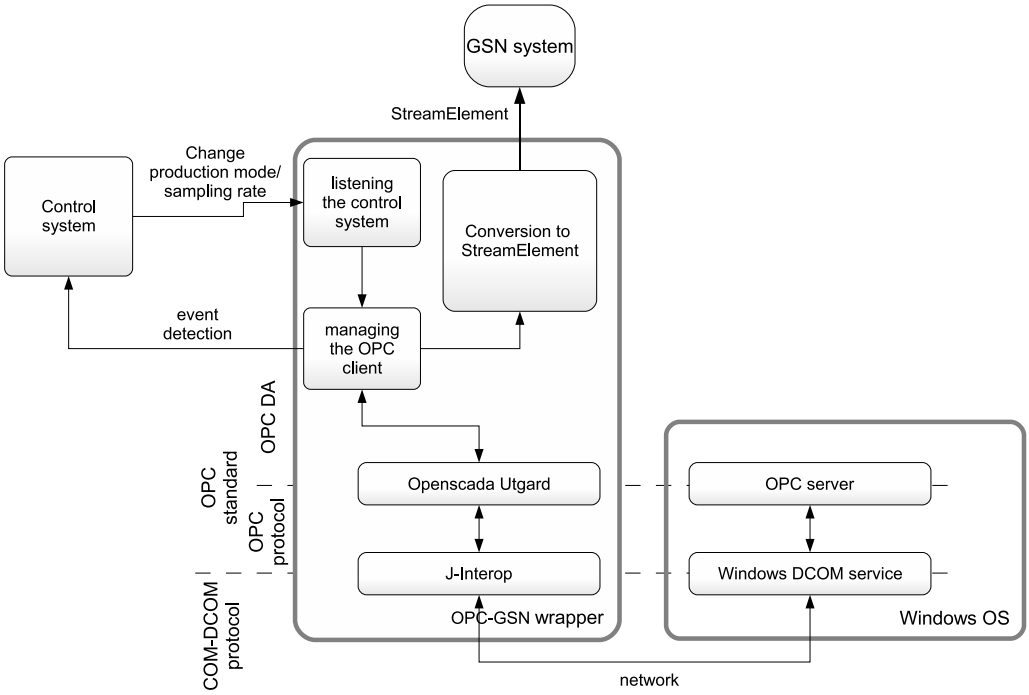


Figure 6: Software architecture of the OPC-GSN wrapper and its environment

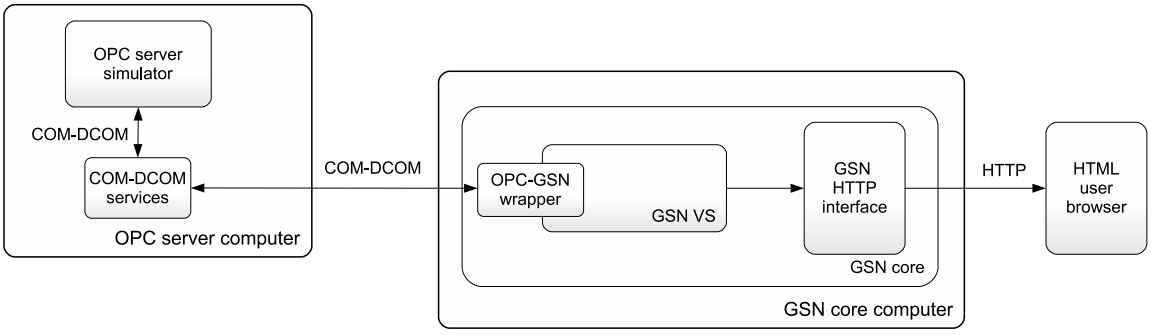


Figure 7: OPC DA compatibility: experiment configuration

5 Experiments

We conducted several tests on our OPC-GSN wrapper to verify three kinds of properties: compatibility of the wrapper StreamElements generated with regard to OPC data types (DA compatibility); expected behaviours of the wrapper in periodic mode (Periodic mode) and in periodic versus change based production modes (Production modes). All experiments run on two computers:

- One computer runs on Microsoft Windows server 2003, and supports an OPC server,
- One computer runs on Red Hat Enterprise Linux AS release 4 and supports the GSN core including our wrapper under test.

During all experiments, we fixed a constant sampling period (Δt_S) on the OPC server and we vary when required, the update period (Δt_U) of the wrapper.

5.1 OPC DA compatibility

The test configuration of the DA compatibility is given on Figure 7. The OPC server is the Matrikon or the DsXp simulator. To check the compatibility of the OPC-GSN wrapper with the OPC DA specification, we generate all possible OPC data types with the OPC simulators. In the GSN core, we deploy a VS that takes StreamElements from the OPC-GSN wrapper and we verify that OPC item values are effectively in the StreamElement generated and with the correct GSN type. We ran the test successively with the two simulators.

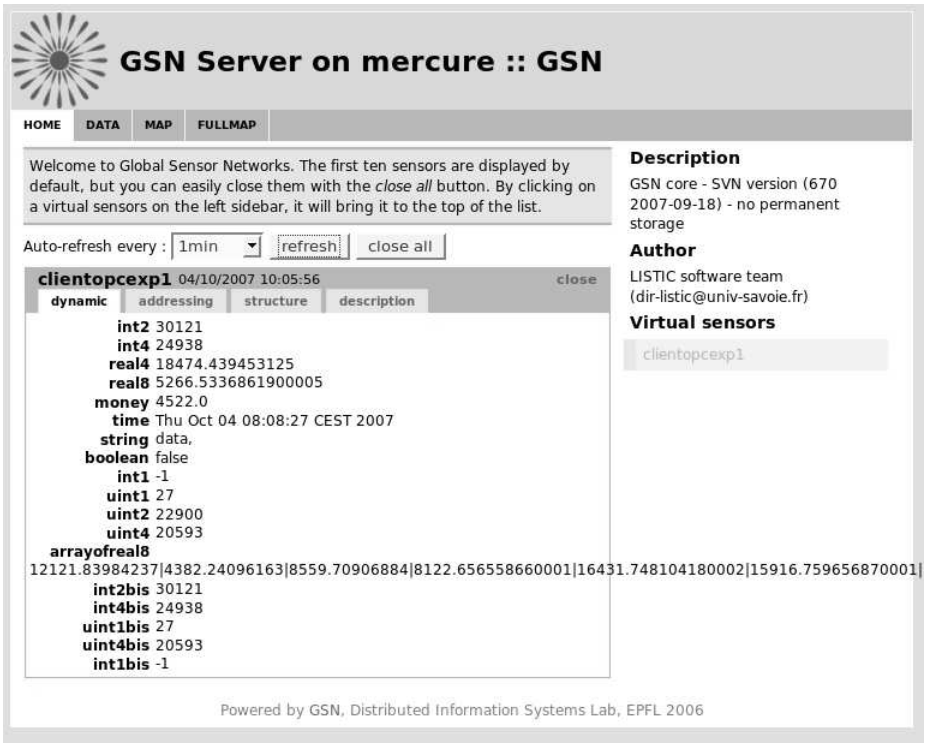
During this test we randomly generate values for items of the types shown in Table 2.

We noticed some limitations with each one. First it is impossible to generate errors on demand with the Matrikon simulator. Another point is the management of the item values. Thus even if it is possible to describe how an item evolve in the Matrikon simulator - by using mathematical expressions - we have note that it is easier in the DsXp Simulator. In the DsXp simulator there are four possible behaviours for the items (constant, ascending, descending and random). These modes are sufficient in our case to test the OPC-GSN wrapper. Concerning DsXp, we have noticed that the OPC data types available is a subset of OPC specification data types. However, it allows for a detailed definition of generated values and their changing dates.

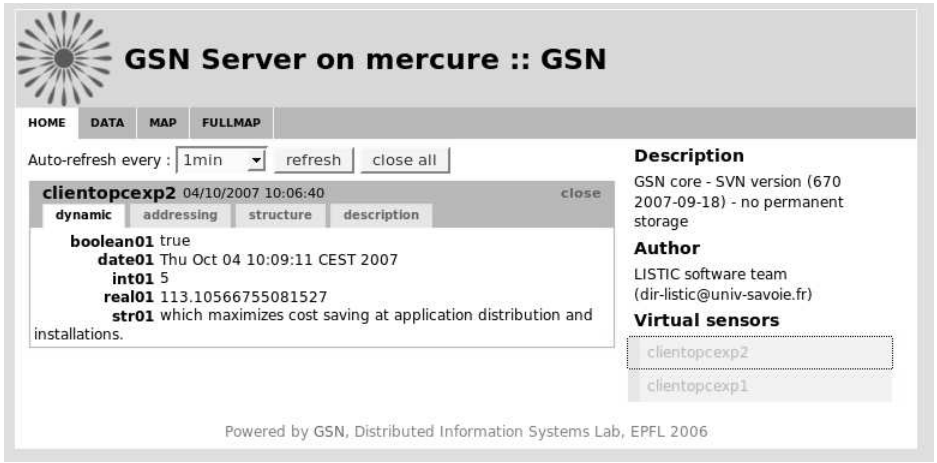
Matrikon During this test we randomly generate values for items of the types shown in Table 2.

The ranges of these data types are taken from the Matrikon simulator which does not generate values on the whole standard ranges.

DsXp During this test we generate values for items of the types shown above (for each OPC type we give the GSN type). The values are generated according to special ranges:



OPC server: Matrikon simulator



OPC server: DsXp simulator

Figure 8: DA compatibility: converted item values generated by the simulators displayed by the GSN WEB interface

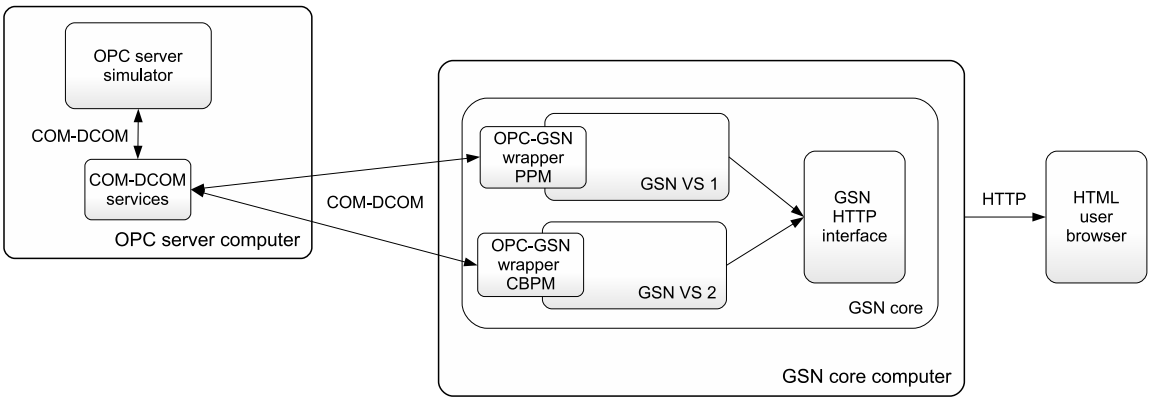


Figure 9: Update rate tests in PPM and two production modes tests: experiment configuration

- *Boolean1* is a boolean converted in a CHAR(5): true and false inversion every second,
- *Date01* is a date converted in a VARCHAR(50): current date and time generated every second,
- *Int01* is an integer converted in an INTEGER: from 10 to 30 with a step of 1 every second,
- *Real01* is a double float converted in a DOUBLE: from 0 to 300 with a step nearly 1 every second,
- *Str01* is a string converted in a VARCHAR(50): cyclic string values changed every second.

All the items are correctly read and can be processed by GSN. Screen shots of the results presented by the Web interface of the GSN core are given in Figure 8

5.2 Periodic production mode

The aim of this test is to verify the timing semantics of the wrapper described in Section 4.1 in periodic production mode. The test configuration is given in Figure 9. The OPC server is the Matrikon or the DsXp simulator. We run two instances of the OPC-GSN wrapper each one connected to a VS. We set $\Delta t_P = \Delta t_U$ at any time: StreamElements are produced as soon as and each time OPC item values are received from the OPC server. One wrapper has a constant update period smaller than the OPC server sampling period ($\Delta t_S > \Delta t_U$). Hence this VS reflects exactly the evolution of the item value in the OPC system with redundant StreamElements. The other wrapper starts with a greater update period than the OPC server sampling period ($\Delta t_S < \Delta t_U$); this is the phase 1. Then, in a second phase, the wrapper uses an update sampling period greater than the OPC sampling period ($\Delta t_S > \Delta t_U$).

In Figure 10, the second VS output is shown on the top of the screen and the first VS output is at bottom. We see that during phase 1, the OPC item value is 1, 2, 3 or 4 (first VS). In contrast, GSN StreamElements (top) provide only values 1 or 4: this exemplifies loss of values due to $\Delta t_S < \Delta t_U$. During phase 2, the first VS (top) correctly produces (redundant) StreamElements with values 1, 2, 3 and 4.

5.3 Production modes

This test checks the behaviour of the wrapper in two production modes: periodic (PPM) and change based (CBPM) as defined in Section 4.3. We set the first wrapper (see Figure 11, top) in PPM with $\Delta t_S > \Delta t_U$. This allows us to see all OPC item values. The second wrapper (see Figure 11, bottom) is running in CBPM.

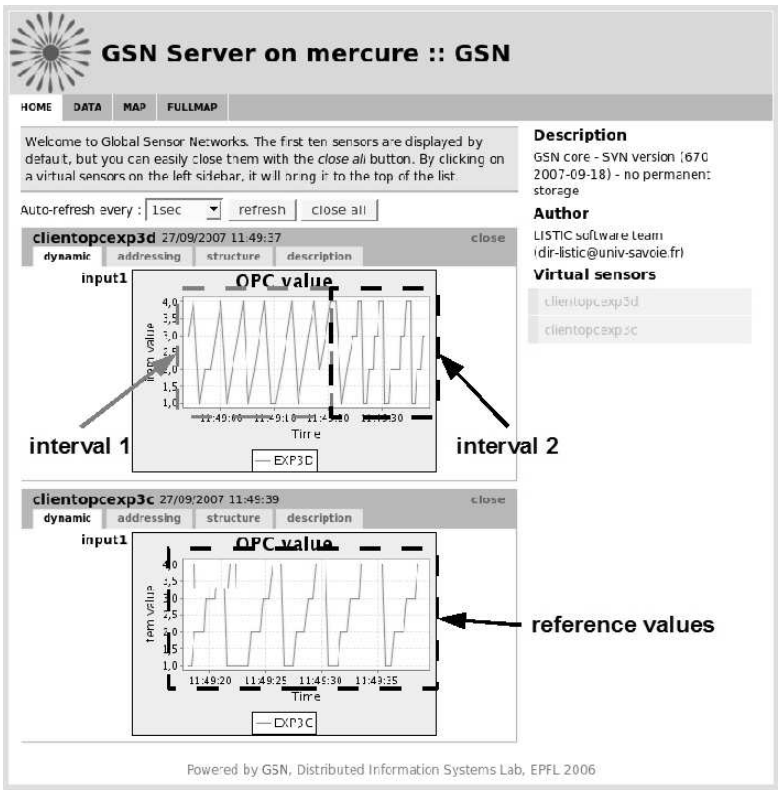


Figure 10: Impact of the wrapper update rate displayed by the GSN WEB interface

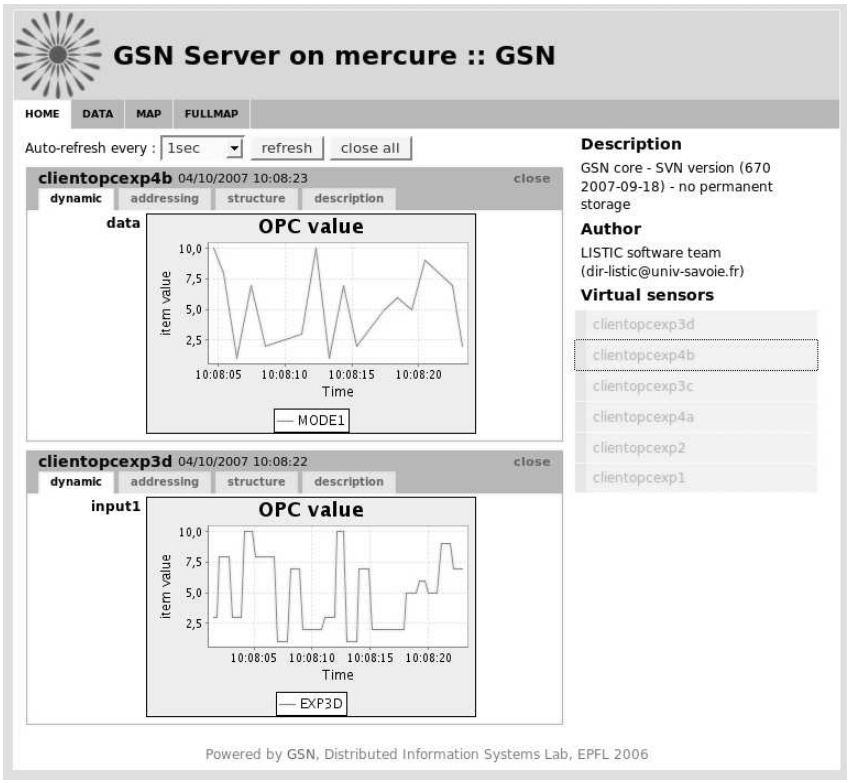


Figure 11: The two production modes displayed by the GSN Web interface

In the PPM, the wrapper produces a StreamElement every update period (i.e. each call to the server). We have set a faster update rate than the sampling rate of the OPC simulator (here DsXp), so the wrapper shows all the values of the item. In the CBPM, the wrapper only produces a StreamElement if the OPC item value, seen by the wrapper, has changed. We can see in Figure 11 (bottom) that the behaviour of the wrapper conforms to the expected one.

6 Conclusion

In this report we have presented design and implementation of a software component, termed wrapper in the GSN context, providing a GSN application with data from an OPC based systems. This work fills the gap between industrial contexts and Information Fusion Systems (IFS): OPC servers are data sources consumed by the IFS built as a GSN infrastructure.

We restricted ourselves to the Data Access 2.05a specification of the OPC foundation to communicate with OPC servers. This is justified first by the large number of OPC servers still using this specification instead of the new OPC foundation Unified Architecture and by the fusion process capabilities of GSN systems such as event detection and historical process.

We also decided to impact as less as possible OPC systems which will be connected to the GSN system. Our solution fits this requirement since there is no modification at all to do on the OPC side of the application using our wrapper. All data processing is supported either by our wrapper or else by GSN features. A noticeable feature of our OPC-GSN wrapper is the ability to set the refreshing period of data read from the OPC servers. This allows the IFS designer to adjust the semantics of the OPC data flow to its fusion process.

Our experiments show that the OPC-GSN wrapper accurately converts all data types of the OPC DA to GSN elementary information unit (stream element) and behave as expected with regard to the refreshing period.

This work is part of our project on adaptable and controlled distributed information fusion systems (ACDIFS). GSN infrastructures allows us to experiment various solutions with regard to the control of connections between fusion nodes and data sources and selection of data sources based on the goals of the ACDIFS. In this respect, we will experiment control mechanisms for connections to data sources (OPC, sensor networks, etc). This will require extended control functionalities of the OPC-GSN wrapper. Another important point is to take into account the new OPC unified architecture(OPC UA). Since OPC UA is based on different communication protocols form the COM/DCOM ones, this extension needs to re-design the communication scheme between OPC servers and GSN.

A OPC-GSN Wrapper User's manual

The OPC-GSN wrapper is a GSN wrapper allowing you to get data from OPC systems into a GSN infrastructure. This manual assumes that you have a running OPC server and a runnable GSN (`http://gsn.sourceforge.net`) core that is to say, the GSN program running in a Java Virtual Machine (JVM).

The OPC-GSN wrapper can convert all OPC data to GSN StreamElement. It can work in Periodic Production Mode (PPM) or in Change based production mode (CBPM). The wrapper asks for OPC item value to the OPC server every Δt_U millisecond. The Δt_U value is stored in a text file (in ms). This file is read regularly by the wrapper so that the wrapper can change its update rate at runtime. You should set $\Delta t_U \leq \min_{i \in I}\{\Delta t_{S,i}\}$, where I is the set of all items converted from the OPC server and $\Delta t_{S,i}$ is the sampling period of item i on the OPC server.

In PPM, the wrapper generates a StreamElement every Δt_U ms. In CBPM, its generates a StreamELement if and only if the last two updated values from the OPC server differ.

A.1 Requirements

On the GSN computer, you will need:

- Java: JRE 1.6 or higher.
- The OPC-GSN wrapper code.
- The Openscada Utgard libraries (`http://openscada.org/UtgardReleases`):
  `openscada-opc-lib-0.x.y.zip` (actually 0.2.0).
  The jar files are in the zip file on the following directory:
  `openscada-opc-lib-0.x.y/lib/*.jar`.

LISTIC libraries The OPC-GSN wrapper is part of the Controlled and Adaptable Distributed Informations Fusion Systems (CADIFUS) project, developed by the Logiciels et Systèmes team of the LISTIC. To allow a full compatibility between the elements produced by the LS team, the following library is required:

- `OPC_GSN_WRAPPER.jar`.

This file can be downloaded from the LS team Web site (`http://www.listic.univ-savoie.fr/ls`) or from the author's personal Web site (`http://olivier.passalacqua.googlepages.com/opcgsnwrapper`).

A.2 Installation

A.2.1 OPC-GSN wrapper

The easiest way to install the OPC-GSN wrapper is to copy the file `OPC\_GSN\_WRAPPER.jar` in the GSN lib directory ({GSNHOME}/lib).

A.2.2 Openscada Utgard

To allow GSN to use the Openscada Utgard project, it is necessary to copy the content of the openscada-opc-lib-0.x.y/lib/ directory in the GSN lib directory ({GSNHOME}/lib).

A.2.3 Registering the OPC-GSN wrapper into GSN

To allow GSN to use our wrapper it is necessary to update the file `$GSNDIR/conf/wrapper.properties` by adding these lines

```
1 ###########################################################
2 # OPC-GSN wrapper - convert OPC values to GSN StreamElement
3 # LISTIC - 2007/10/01
4 wrapper.name=opcWR
5 wrapper.class=fr.univ.savoie.listic.sysrepwrappers.OPC.ClientWR
```

Important: do not modify the name of the wrapper in line 4.

A.2.4 OPC-GSN parameters files

The OPC-GSN wrapper uses two text files to tune during its execution the production mode and the update period. These two files can be created anywhere in the GSN directory tree. It is only necessary to know their name and where they are to report this information in the XML description. We consider these two files are named:

- pm.txt for the production mode,
- wup.txt for the wrapper update period.

It is important to only write the value of the considered attribute in each text file.

A.2.5 Checking network properties

The network configuration should allow the OPC server and the wrapper on the GSN computer to communicate with COM-DCOM protocols. This may require to check the security settings of the network equipments (firewalls for instance). You should especially be aware of:

- The system software used by any OPC server is allowed to communicate:
  `C:\WINDOWS\system32\Opcenum.exe`,
- In the case of Matrikon and DsXp simulators, these programs are allowed to communicate:
  `C:\Prog...\DSxP\DSxPOpcSimulator\DSxPOpcSimulator.exe`,
  `C:\Prog...\Matrikon\OPC\Simulation\OPCSim.exe`,
  `C:\Prog...\..\MatrikonOPC\Common\PSTCFG.exe`.
- In the case of production servers, the core program may already be allowed to communicate.

A.3 Running a GSN core with the OPC-GSN wrapper

A.3.1 Classpath and GNS launching

In every run, the Openscada library jar files and the OPC-GSN wrapper jar file must be in the Java class path. Moreover, Openscada library requires `j-interop.jar` to be the first in the Java class path.

Before running the OPC-GSN wrapper, it is important to modify the script that constructs the Java class path. We give two ways to do this according to how the GSN core is executed.

***bash* script** This script is automatically generated during the installation of previous versions of GSN. If you are using the SVN version please go to the next paragraph.

```
1 #!/bin/bash
2 cp=$GSNDIR/lib/j-interop.jar
```

```
cp=$cp:/usr/lib/jvm/java-6-sun-1.6.0.00/jre/../lib/tools.jar
for jarFile in $( ls lib/*jar ); do
  cp=$cp:$GSNDIR/$jarFile
done
java -classpath $cp gsn.Main
```

***ant* script** This script is in the `build.xml` file of GSN. The modifications are in the *<path...>* element description.

```
<path id="classpath">
        <pathelement location="lib/j-interop.jar"/>

        <pathelement location="${build.dir}/jcoverage-classes/"/>
        <pathelement location="${build.dir}/"/>
        <pathelement location="${env.JAVA_HOME}/lib/tools.jar"/>
        <fileset dir="${libdir}">
           <include name="**/*.jar"/>
        </fileset>
      </path>
```

A.3.2 Using the OPC-GSN wrapper inside a Virtual Sensor

A wrapper is always used inside a Virtual Sensor (VS) in GSN. So all the attributes of the OPC-GSN wrapper - such that the address of the server or the items names - are defined in the XML description of a VS. When GSN activates this VS, the wrapper is set up and can connect the OPC server. The description of such a VS is given bellow in a dedicated section and must be stored in the virtual-sensor directory of GSN.

A.4 OPC-GSN wrapper behaviour

A.4.1 Production modes and update period

The user of the OPC-GSN wrapper should define the refresh period of the wrapper. This period defines the time in millisecond between two calls from the wrapper to the OPC server: this period is noted Δt_U. During its execution, the wrapper read the value of Δt_U presents in a text file - the fie's name is indicated as an attribut of the wrapper in a VS description (see the listing in Section A.5 for an example of description). When the wrapper has updated the local item value, it can produce a StreamElement following modes:

- periodic production mode (PPM): in this mode the wrapper produces a StreamElement each Δt_U msec.
- change based production mode (CBPM): in this mode the wrapper compares the local value of an item before its update and after its update. If the value or the quality has changed, the wrapper produces a StreamElement.

To manage the production mode, the user of the wrapper should indicate the name of the text file in which the production mode is noted (PPM or CBPM). This file is read during the update period of the wrapper and before the production of the StreamElement.

A.4.2 OPC to GSN data type mapping

OPC data type	Java data type	GSN data type
short integer	Short	SMALLINT
integer	Integer	INTEGER
single float	Double	DOUBLE
double float	Double	DOUBLE
currency	Double	DOUBLE
date	String	VARCHAR(100)
string	String	VARCHAR(100)
boolean	String	CHAR(5)
byte	Byte	TINYINT
unsigned byte	Short	SMALLINT
unsigned word	Integer	INTEGER
unsigned double word	Long	BIGINT
array of double	String	VARCHAR(1000)
word	Short	SMALLINT
double word	Integer	INTEGER
unsigned short	Integer	INTEGER
unsigned integer	Long	BIGINT
character	Short	SMALLINT

Table 3: Data type mapping between OPC, Java and GSN

A.4.3 OPC-GSN wrapper StreamElement format

The OPC-GSN wrapper creates a GSN StreamElement based on the OPC values following this format:

- vaitem*num* (*GSNtype*) is the value of the num^{th} item in the XML description and its *GSNtype* is automatically computed from the OPC data type,
- ecitem*num* (INTEGER) is the error code of any detected error,
- quitem*num* (INTEGER) is the quality of the num^{th} item,
- tsitem*num* (BIGINT) is the timestamp of the num^{th} item.

An example of how to use the StreamElement is given in the requests of the following VS description.

A.5 OPC-GSN wrapper and Virtual Sensors

A wrapper is always used by a virtual sensor (VS), so we describe the attributes of the OPC-GSN wrapper in a VS which will convert the OPC information to the GSN format and processes data. We provide below an example description file corresponding to the configuration:

- One GSN core running on a computer named *mercure*,
- One OPC server (Matrikon simulator) running on a computer and its IP address is 192.xxx.xxx.xxx,
- Two items are read on the server and we want to show their values and the quality of the first item and the timestamp of the second one,

- The access path to the 2 text files are:
  {GSNHOME}/userconf/pm1.txt
  {GSNHOME}/userconf/wup1.txt

. Details are given below.

```
<virtual-sensor name="opcconverter" priority="10" >
<!--
 Data access test:
 server used : Matrikon
 PM file : userconf/pm1.txt
 WUP file : userconf/wup1.txt

 items names and types :
 item0 : Random.Int2
 type0 : short integer
 item1 : Random.Int4
 type1 : integer
-->
 <processing-class>
  <class-name>gsn.vsensor.BridgeVirtualSensor</class-name>
   <init-params>
   </init-params>
  <output-structure><!-- the data produced by the VS -->
   <field name="Int2" type="SMALLINT" />
   <field name="QualityInt2" type="INTEGER" />
   <field name="Int4" type="INTEGER" />
   <field name="TimeStampInt4" type="BIGINT" />
  </output-structure>
 </processing-class>

 <description>
   OPC-GSN wrapper - connected to Matrikon OPC simulator - 2 items read
 </description>

 <life-cycle pool-size="3" />

 <addressing>
  <predicate key="geographical">mercure</predicate>
 </addressing>

 <storage history-size="50"/>

 <streams>
  <stream name="input1">
   <source alias="server1" storage-size="1" sampling-rate="1">
    <!-- description of the OPC-GSN wrapper -->
    <address wrapper="opcWR"><!-- same name as in the configuration file -->
     <predicate key="warn">false</predicate><!-- warning mode -->
     <!-- OPC server information -->
     <predicate key="host">192.xxx.xxx.xxx</predicate><!-- OPC server address -->
     <predicate key="domain">localhost</predicate><!-- OPC server domain -->
     <predicate key="user">username</predicate><!-- user allowed on the server -->
     <predicate key="password">userpassword</predicate><!-- user password -->
     <!-- Matrikon DCOM ID-->
     <predicate key="classId">F8582CF2-88FB-11D0-B850-00C0F0104305</predicate>

     <!-- wrapper attributes -->
     <!-- text file where the refresh period is-->
     <predicate key="wup">userconf/wup1.txt</predicate>
     <!-- text file where the production mode is-->
     <predicate key="pm">userconf/pm1.txt</predicate>

     <!-- name of the items group on the OPC server -->
     <predicate key="group">gsn_exp4b</predicate>

     <!-- names and types of the items -->
     <predicate key="item0">Random.Int2</predicate><!-- id first item -->
     <predicate key="type0">short integer</predicate><!-- OPC type first item -->
     <predicate key="item1">Random.Int4</predicate><!-- second item -->
     <predicate key="type1">integer</predicate><!-- etc -->
    </address>
    <!-- end of the OPC-GSN wrapper description -->

    <query>SELECT * FROM wrapper </query> <!-- first level request -->

   </source>
   <query><!-- global request -->
    SELECT
```

```
74        vaitem0 as Int2,
75        quitem0 as QualityInt2,
76        vaitem1 as Int4,
77        tsitem1 as TimeStampInt4
78      FROM
79        server1
80      </query>
81    </stream>
82  </streams>
83  </virtual-sensor>
```

Lines 14 to 24 - Processing class Since we simply transfer data, we only use the bridge VS given by GSN.

Lines 43 to 50 - OPC client configuration We set here the attributes of the (Openscada) OPC client. If the warning mode is activated, information given in this verbose mode are written in the log file of GSN. The host predicate must be the IP address of the server and not its hostname.

Lines 52 to 56 - OPC-GSN wrapper configuration The <*wup*> predicate gives the name of the text file with the update period value (from the OPC server to the wrapper) expressed in msec. The <*pm*> predicate gives the name of the text file with the production mode value: PPM for period production mode and CBPM for change based production mode. In the PPM a StreamElement is produced every period, and in the CBPM a StreamElement is produced only if the value or the quality of at least one item has changed.

Lines 61 to 65 - Items configuration These lines describe the items read on the OPC server. The predicate *item* must give the same id as the one registered on the OPC server. The predicate *type* must be one of the OPC data types (see table 3).

Lines 69 to 80 - SQL requests We can select and aggregate values and/or quality of the items. The corresponding fields in the StreamElement are:

- vaitem for the value of an item,
- ecitem for the error code if any error,
- quitem for the quality of the value,
- tsitem for the timestamp of the item, given by the OPC server.

References

[Ali07] Ali Salehi. *The GSN book*. 14 April 2007. http://gsn.svn.sourceforge.net/.

[CCJ06] Christian Niekerke, Christoph Schmidtbauer, and Jens Reimann. OpenSCADA project. http://openscada.org, 2006.

[KMA07a] Karl Aberer, Manfred Hauswirth, and Ali Salehi. Infrastructure for data processing in large-scale interconnected sensor networks. In *Mobile Data Management (MDM)*, 2007.

[KMA07b] Karl Aberer, Manfred Hauswirth, and Ali Salehi. Zero-programming sensor network deployment. In *Applications and the Internet Workshops, 2007. SAINT Workshops 2007. International Symposium on*, 15 January 2007.

[Li 02] Li Zheng. OPC (OLE for process control) specification and its developments. In *SICE 2002. Proceedings of the 41st SICE Annual Conference*, 5 August 2002.

[OPC] OPC foundation. http://www.opcfoundation.org/.

[Ros98] Rosemary Rock-Evans. *DCOM Explained*. Digital Press, 1998.

[SW06] Stefan-Helmut Leitner and Wolfgang Mahnke. OPC UA - Service oriented Architecture for Industrial Applications. In *Outstanding Retailer Awards*, October 2006.